ARéVi : une boîte à outils 3D pour des applications coopératives

T. Duval[1], S. Morvan[2], P. Reignier[2], F. Harrouet[2], J. Tisseau[2]

[1] : *ENSAI, Campus de Ker Lann, F 35170 Bruz*
thierry.duval@ensai.fr
[2] : *ENIB, LI[2], Technopole Brest Iroise, CP 15, F 29608 Brest cedex*
[morvan, reignier, harrouet, tisseau]@enib.fr

RÉSUMÉ. *ARéVi est une nouvelle boîte à outils de réalité virtuelle distribuée. Son noyau (un ensemble de classes C++) permet de créer des applications de réalité virtuelle coopératives et distribuées, et ce sans programmation. Cette boîte à outils est largement ouverte, ce qui permet de créer des applications évoluées comme des simulations de systèmes multi-agents intelligents. Elle est particulièrement adaptée à la création d'applications coopératives : plusieurs applications (que nous appellons des "sessions ARéVi") peuvent partager un même univers 3D, de façon à ce que plusieurs utilisateurs puissent coopérer en interagissant avec les mêmes objets. Dans cet article, nous présentons essentiellement les caractéristiques ccopératives d'ARéVi, ainsi que les possibilités d'évolution dynamique de programmes, essentielles pour la coopération.*

ABSTRACT. *ARéVi is a new distributed virtual reality toolkit. Its kernel (a set of C++ classes) allows to create distributed virtual reality applications without programming. This toolkit is widely open, which allows the creation of evoluated applications like intelligent multi-agents systems simulations. It is particularly adapted for creating collaborative software : several "applications" (what we call "ARéVi sessions") can share a common 3D universe, so several users can interact together with the same objects. In this paper, we only discuss of the collaborative features of ARéVi, and of the facilities provided by ARéVi for program dynamic evolution, essentials to cooperation.*

MOTS-CLÉS : *Réalité virtuelle distribuée, Simulations 3D, Systèmes multi-agents, Boîte à outils de réalité virtuelle, Applications coopératives, Collecticiel, Evolution Dynamique de Programme*

KEYWORDS: *Distributed Virtual Reality, 3D Simulation, Multi-Agents Systems, Virtual Reality Toolkit, CSCW, Groupware, Program Dynamic Evolution*

1. Introduction : qu'est-ce qu'ARéVi ?

ARéVi (**A**telier de **Ré**alité **Vi**rtuelle) est une plate-forme, multi-processus, de développement d'applications de réalité virtuelle distribuées. Ecrite en C++, elle possède un noyau indépendant des bibliothèques de rendu 3D. Cette boîte à outils peut être utilisée pour développer rapidement, sans programmation, des applications de visualisation 3D "temps réel", avec ou sans immersion des opérateurs humains, des applications de travail coopératif ou des applications de réalité virtuelle distribuées intégrant des univers peuplés d'agents au comportement simple.

L'extension des classes de base, par héritage ou composition, permet un niveau supplémentaire de développement, par exemple des simulations d'agents intelligents ou des jeux distribués.

Les deux aspects principaux d'ARéVi sont la distribution des agents et la coopération entre agents. ARéVi permet par exemple de créer des simulations distribuées où les différents agents en présence sont répartis sur différents nœuds d'un réseau, ceci permet en particulier :

- de répartir les calculs des comportements des agents sur plusieurs machines,
- de faire coopérer plusieurs agents humains, chacun aux commandes d'une application particulière.

La coopération entre agents "informatiques" est par ailleurs possible, que les agents soient répartis ou non.

Sur chacun des sites participant à une session distribuée/coopérative ARéVi, on peut faire évoluer des agents "réels" (calculs complexes effectués sur le site) au milieu de copies/reflets/fantômes (calculs simplifiés) d'agents situés sur les autres sites de la session.

Avec un exemple simple de proies et de prédateurs évoluant dans un même univers, il est possible :

- de répartir les proies et les prédateurs sur différentes machines, ceci afin de répartir la charge de calcul entre plusieurs sites,
- de comparer les "intelligences" des différents agents, en particulier il est possible de mettre un agent humain aux commandes d'un agent afin d'éprouver les algorithmes des autres agents et/ou d'éprouver les ressources de l'humain.

Une application ARéVi peut joindre ou quitter une simulation collective à tout moment, apportant ou emportant avec elle ses propres agents. En quittant une simulation, elle peut éventuellement en "faire don" aux autres applications participant à la simulation, et de façon symétrique, elle peut héberger des agents en provenance d'autres applications, parce que ces dernières quittent la simulation ou bien pour répartir les charges de calcul.

Enfin, certaines applications peuvent être simplement à l'écoute de ce qui se passe sur le réseau, ne faisant ainsi évoluer que des "fantômes" d'agents, sans que les autres applications d'une session coopérative en soient informées. De telles applications sont adaptées à la visualisation des agents.

Notre étude se situe dans le cadre d'un nombre important de recherches visant à l'élaboration de boîtes à outils pour la réalité virtuelle et la simulation distribuée (MR Toolkit [6], VR-DECK [4], DIVE [3], NPSNET/DIS [2], [9]), ou l'animation et l'interaction 3D (UGA [12], VB-II [5], Virtual Studio [1]).

En parallèle à ces études, depuis ces dernières années, un effort important a été réalisé, par les constructeurs et les éditeurs, pour améliorer les algorithmes de rendu et élaborer des moteurs graphiques efficaces. Certains produits commerciaux sont présents sur le marché : World Toolkit (Sense8, 1991), dVISE (Division) ou Clovis (MEDIALAB, 1995).

En normalisation, nous assistons à l'émergence d'un standard de description d'applications de réalité virtuelle : V.R.M.L. 2.0 [ISO/IEC WD 14772], [8], même si les aspects comportementaux ou distribués ne sont pas encore réellement pris en compte. Coté communication, la prochaine version du protocole IP (IP v6) [RFC 1883 (décembre 1995)] introduit la notion de flot et inclut en natif le multipoint, entre autres.

De ce fait, une proposition pour une nouvelle architecture de boîte à outils pour la réalité virtuelle se doit d'être très largement ouverte et le plus possible indépendante des bibliothèques de rendu pour pouvoir utiliser les solutions les plus optimales [10], [11], elle devra pouvoir décrire des environnements de plus en plus larges, comportant des agents aux comportements de plus en plus complexes.

2. Architecture logicielle d'ARéVi

Le noyau d'ARéVi est constitué de plusieurs classes C++, dont les deux principales sont l'entité et l'application : l'application est un réceptacle d'entités, elle les fait évoluer et leur permet de communiquer.

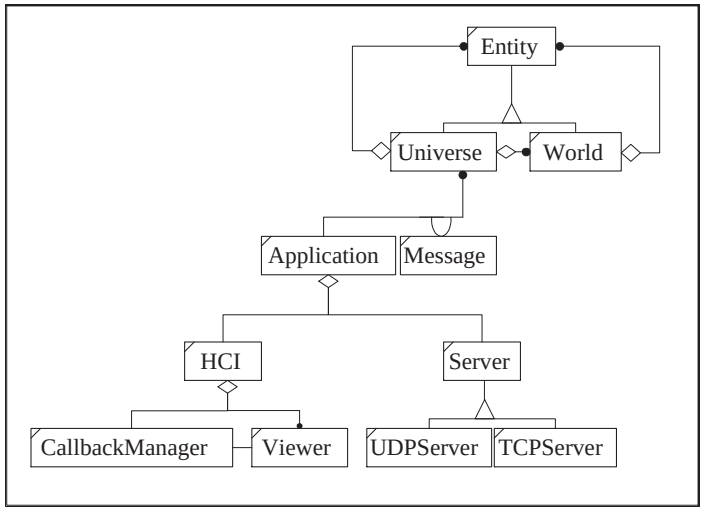

FIG. 1 – *Les classes C++ du noyau d'ARéVi*

L'entité est la classe de base des agents qui évolueront dans nos applications.

2.1. *Des sessions coopératives*

Nos propres applications sont réparties sur les nœuds d'un réseau :

– local : elles communiquent alors en point à point et/ou en diffusion,

– distant : elles communiquent seulement en point à point.

Elles permettent aux entités de communiquer entre elles, et elles fournissent des renseignements sur les entités qu'elles gèrent.

Elles fonctionnent selon une architecture client/serveur : elles offrent des fonctionnalités permettant d'accéder aux entités qu'elles hébergent, et sont elles-mêmes utilisatrices des services des autres applications participant à une session de travail coopératif.

Ces services sont offerts par un serveur TCP (en connexion point à point, pour des communications entre réseaux distants) et par un serveur UDP (en connexion point à point ou en diffusion, pour des communications sur un réseau local).

2.2. *Des entités et des fantômes*

Les entités sont les agents de base qui évoluent dans nos applications, ces agents peuvent être réels ou bien être des fantômes.

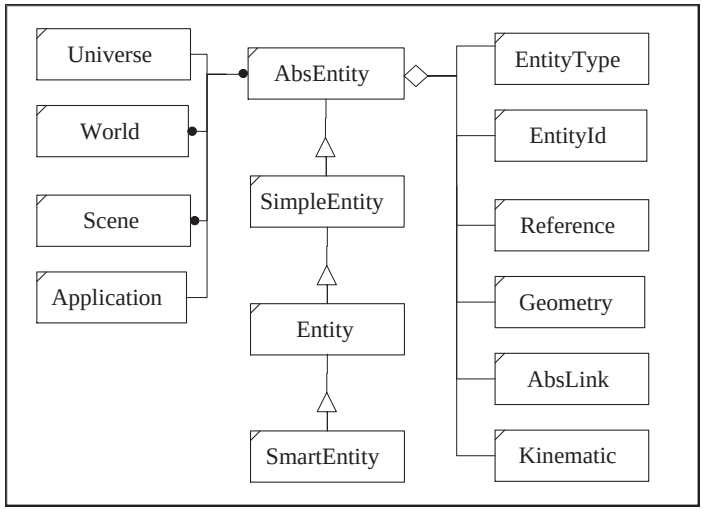

FIG. 2 – *Entités : principales caractéristiques*

Les entités ont les caractéristiques suivantes (voir figure 2) :

– un identifiant unique (date, hostame, alias, clé),

– des référentiels local et global,

– une géométrie 3D (ou plusieurs, pour des animations et/ou des niveaux de détail),

– des contraintes de liaisons (dont mécaniques),

– une position, une orientation, des vitesses et accélérations (linéaires et angulaires).

Les entités réelles possèdent un comportement évolué, elles sont susceptibles d'être reflétées dans d'autres applications (sous la forme de fantômes) lorsqu'elles sont partagées entre plusieurs applications.

Les fantômes ont un modèle comportemental simplifié, qui nécessite peu de calculs. Pour le moment, ce modèle est un modèle cinématique (coordonnées, vitesse et accélération, en position et en orientation)

2.3. *Des agents*

Un agent dérive d'une entité, il se comporte selon son propre modèle, qui peut être implémenté en logique floue, à l'aide de réseaux de neurones ou de réseaux de Petri. On peut trouver des fantômes d'agents dans d'autres applications.

Les agents peuvent être des lumières, des caméras, des proies, des prédateurs, des points de vues, des explorateurs, ... (voir figure 3).

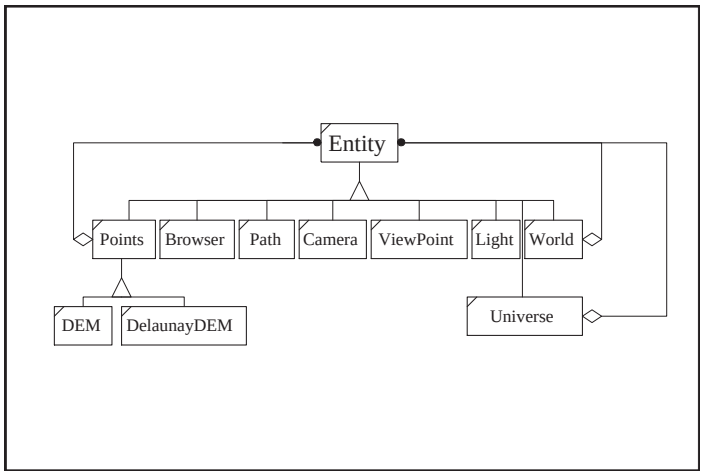

FIG. 3 – *Les agents sont des spécialisations d'entités*

3. Comportement des entités

3.1. *Utilisation de modèles prédictifs*

Les entités réelles ont connaissance de ce modèle utilisé par les fantômes, ce qui leur permet de faire du “dead-reckoning” : elles ne diffusent leur état à leurs fantômes potentiels que si cet état est significativement différent de celui que les fantômes peuvent anticiper. Ceci dans le but de réduire le trafic sur le réseau.

A la création d’une entité réelle partagée, une application ARéVi va diffuser cette création aux autres applications participant à la session coopérative, en leur demandant de créer un fantôme. Ce fantôme va évoluer grâce à son modèle comportemental simplifié. Actuellement, ce modèle simplifié décrit plus haut est “codé en dur”, il ne peut pas évoluer au cours du temps, et il est le même pour tous les fantômes, quel que soit le type de l’entité réelle associée. Nous sommes en train de le coder à l’aide d’un langage qui pourra être interprété par nos applications, ce qui permettra de le transmettre (en totalité ou en partie) aux fantômes lors de leur création.

Lors d’une constatation de décalage entre l’état réel de l’entité et l’état du modèle prédictif des fantômes, l’entité émet une demande de mise à jour vers ses fantômes potentiels.

Les applications concernées interprètent cette requête et la transmettent aux fantômes correspondants.

De même, lors de la destruction d’une entité réelle partagée, l’application qui l’héberge diffuse aux autres applications de la session une demande de destruction des fantômes associés à cette entité.

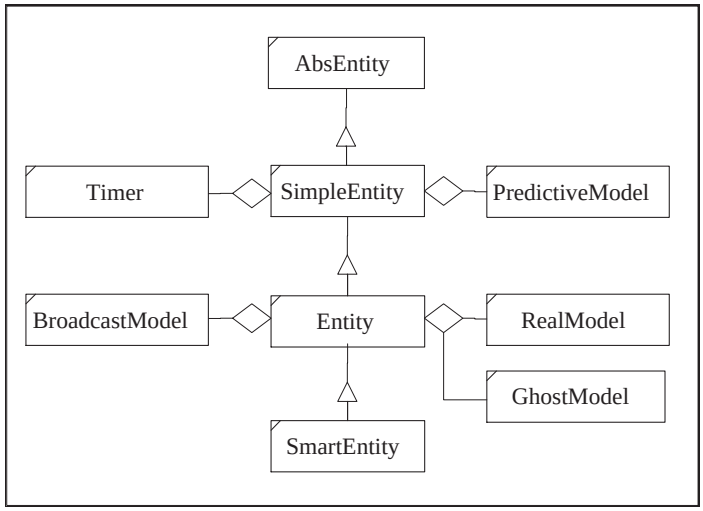

FIG. 4 – *Entitiés : diffusion et comportement*

3.2. *Les aspects réseaux*

Le mode de diffusion que nous utilisons actuellement est le mode “broadcast” des sockets UNIX UDP : nos applications coopératives doivent donc être impérativement sur le même réseau local. Nous sommes en train de passer à un mode de diffusion à des abonnés (sockets UNIX UDP en mode “multicast”) qui nous permettra des sessions coopératives à travers internet. Il ne nous reste actuellement qu’à nous raccorder au FMbone.

Le fait d’utiliser une transmission par datagramme peut causer des pertes d’information sur le réseau. Il peut donc arriver qu’une demande de création, de mise à jour ou de destruction de fantôme se perde dans la nature. Pour pallier ce problème, nous avons mis en œuvre les mécanismes suivants :

- en cas de réception d’une demande de mise à jour d’un fantôme inconnu, une application va demander à l’application émettrice de lui envoyer des informations complémentaires en point à point TCP (fiable). Si c’est le premier message en provenance de cette application (cas où l’application réceptrice vient d’entrer dans la session coopérative), on lui demandera de fournir des informations concernant toutes ses entités réelles (pour éviter une multiplication de messages), sinon (cas d’une demande de création égarée) on lui demandera seulement de fournir des informations sur l’entité à créer. Dans le cas d’une nouvelle arrivée sur le réseau, on choisit de ne demander des informations sur les seules entités réelles plutôt que sur la totalité des entités afin de répartir la charge de communication sur toutes les applications de la simulation possédant des entités réelles.

- en cas de perte de demande de mise à jour de fantômes, le comportement des fantômes peut significativement dévier de celui estimé par l’entité réelle associée. Ceci risque de perdurer si l’entité réelle ne subit pas, durant une période importante, de nouvelles modifications ne pouvant être prédites par le modèle des fantômes. Pour limiter ces risques de persitances de décalages, les entités émettent régulièrement leur état, même s’il n’a pas significativement évolué depuis leur dernière émission. Bien sûr, cet envoi systématique se fait à une fréquence moindre que celle à laquelle évoluent les entités.

- en cas de perte d’une demande de destruction d’entité, il pourrait y avoir des fantômes résiduels dans certaines applications. Pour remédier à ce problème, les fantômes n’ayant pas reçu de mise à jour depuis trop longtemps sont supprimés par l’application dans laquelle ils évoluent.

3.3. *Interactions avec les entités*

Lors d’une interaction de l’utilisateur ou d’une entité avec une autre entité, les choses vont se passer différemment selon que cette dernière est une entité réelle ou bien un fantôme :

- une entité réelle va traiter l’interaction,

– un fantôme va envoyer une requête à son entité responsable ou bien traiter l'interaction s'il en est capable (on parle alors de délégation sémantique locale).

S'il n'est pas capable de traiter l'interaction, un fantôme envoie donc un message à son entité responsable : ce message peut ne pas attendre de réponse (par exemple envoi d'une demande de déplacement) ou bien en attendre une (par exemple demande de renseignement que le fantôme ne sait pas satisfaire). Dans ce dernier cas, l'entité réelle recevra un message auquel elle devra répondre. Ces échanges bidirectionnels entre une entité réelle et l'un de ses fantômes se font en point à point en mode TCP, pour des liaisons fiables.

Ces échanges d'informations entre entités rélles et fantômes peuvent être relativement lourds et donc ralentir les interactions. Aussi, il est parfois souhaitable de donner une certaine autonomie aux fantômes. Actuellement, cette autonomie doit être prévue a priori, elle ne peut être que partielle et temporaire pour des fantômes d'agents évolués. Seules les entités de base peuvent fournir une autonomie totale à l'un de leurs fantômes, c'est absolument impossible pour les agents évolués qui ne font pas partie du noyau d'ARéVi, dont toutes les fonctionnalités ne peuvent donc pas être connues a priori.

Grâce à la nouvelle version d'ARéVi intégrant oRis (voir section suivante), le comportement des entités peut être modifié dynamiquement, pour une entité particulière ou pour un groupe d'entités. Celui des fantômes également. Cela peut permettre de tester de meilleurs modèles prédictifs (de façon interactive) ou bien de transmettre de nouveaux modèles de comportement.

Il sera donc possible de doter dynamiquement des fantômes de responsabilités, partielles ou totales, et ce même pour des agents évolués.

Cela permettra également des migrations temporaires ou définitives, partielles ou totales, d'agents d'une application vers une autre.

3.4. *Migrations d'agents*

Il est essentiel de permettre à nos agents de migrer d'une session ARéVi vers une autre, et ce pour plusieurs raisons :

– pour répartir dynamiquement la charge de calcul au travers du réseau,

– pour des interactions homme-machine plus performantes,

– pour récupérer certaines entités évoluant dans une session ARéVi quittant une simulation,

– pour remplacer des fantômes par des entités réelles en cas de "crash" d'une ou plusieurs sessions ARéVi.

Ces migrations ne peuvent être obtenues que s'il est possible de faire migrer le comportement de nos entités et agents au travers du réseau, à l'aide d'un langage adapté et de l'interpréteur correspondant dans nos applications ARéVi.

C'est ce dernier point qui va être discuté dans la section suivante.

3.5. *Remarque*

Le terme “fantôme” employé ici est à rapprocher des “reflets” ou des “proxys” également employés dans la littérature.

4. Programmation sous ARéVi

Comme tout système de réalité virtuelle, ARéVi est basé sur un langage permettant non seulement de construire les univers virtuels (indiquer les objets présents et leur comportement) mais permettant également de préciser des éléments d’interface (type et position des fenêtres, etc).

Parmi les langages existants, nous avons retenu *oRis* [7]. Ce langage est développé au sein de notre laboratoire pour l’étude des systèmes multi-agents. Il possède les caractéristiques nécessaires à nos besoins :

- langage agent interprété,
- langage dynamique,
- persistance des agents,
- peut être “immergé” au cœur d’une application C ou C++.

Nous allons revenir maintenant plus en détail sur ces différents aspects.

4.1. *Présentation générale de l’architecture*

Le concept fondamental du langage oRis est celui d’agent. Un agent comporte deux aspects :

1. une structure de donnée. Cette structure reprend les principes objets. Elle permet le stockage d’attributs et de méthodes. Elle intègre la notion d’héritage simple ou multiple. La syntaxe est proche du `C++`.
2. un comportement autonome. Ce comportement existe si l’agent possède une méthode particulière : la méthode `main(void)`. Il est alors automatiquement et périodiquement activé par le “schéduleur” du langage.

Nous avons choisi de représenter tous les éléments manipulés par ARéVi sous la forme d’agents. Cela comprend donc non seulement les objets “physiques” faisant partie de l’univers virtuel mais également les structures de données nécessaires à la mise en place de la simulation.

Le principal avantage de représenter les structures de données de base d’ARéVi sous forme d’agent est que l’on peut très facilement par dérivation leur donner un comportement. Par exemple, l’agent `ArScene` n’est constitué de base dans ARéVi que d’une liste d’agents visualisés lorsque l’on regarde dans cette scene. On peut maintenant créer une `ProxyScene`, dérivée d’une ArScene et fournissant un comportement au travers une méthode `main()`. Ce comportement peut être par exemple d’intégrer automatiquement à la scène tout objet virtuel pénétrant dans la zone géographique

concernée. De même, un viewer ARéVi (fenêtre de visualisation 3D) possède comme attribut le nombre d'images générées par seconde. On peut très facilement créer un viewer dont le comportement est de dégrader la qualité de son affichage si ce nombre d'images tombe en-dessous d'un certain seuil.

4.2. *Interaction*

L'interaction entre les agents et les opérateurs humains au sein de nos univers virtuels se situe à plusieurs niveaux : interaction clavier, envoi de messages, et plus généralement envoi de scripts.

4.2.1. *Interaction clavier*

L'interaction la plus simple et la plus directe sous ARéVi est l'interaction grâce aux boutons de la souris ou grâce aux touches du clavier. Par l'intermédiaire de la librairie Open-Inventor, il est possible de connaître le nom de l'agent présent sous le curseur de la souris lorsqu'une touche du clavier est enfoncée. Si cet agent dérive de la classe ArEvent, il est alors sensible à cet événement et sa méthode dont le nom est celui de la touche précédé de `event` est appelée. Si cette méthode n'est pas définie au niveau de l'agent, c'est alors celle de ArEvent qui est considérée. Cette méthode par défaut ne contient pas de code.

4.2.2. *Communication par scripts*

oRis est un langage interprété. Il est possible à tout moment d'intervenir sur le programme en cours d'interprétation sans que pour cela il soit nécessaire d'interrompre la simulation. Cette intervention consiste en un script oRis interprété par ARéVi. Ce script peut être envoyé par les différents intervenants au moyen de liens TCP. Ils peuvent être directement saisis au travers une interface texte ou, pour une interface plus traditionnelle, associés à des actions de type sélection d'un item dans un menu ou pression d'un bouton.

Le code qu'un utilisateur peut envoyer à une session ARéVi est n'importe quel code oRis. Tout ce qu'un agent peut faire, un utilisateur peut également le faire. Ceci permet une interaction très précise et très profonde avec le système donnant une très grande liberté d'action dans nos univers virtuels.

4.3. *Aspects dynamiques du langage*

oRis est un langage agent interprété offrant la possibilité en cours d'exécution :

- de créer de nouvelles classes,
- de redéfinir des méthodes de classe,
- de redéfinir des méthodes d'instance.

Les deux premiers aspects se retrouvent dans d'autres langages interprétés ou semi-interprétés comme Java. Il permet à tout intervenant d'apporter ses propres modèles d'agents et de pouvoir les inclure dynamiquement dans un univers ARéVi en cours de fonctionnement sans avoir à l'arrêter au préalable.

Le dernier point (redéfinition de méthodes d'instance) est une des particularités originales de notre langage. Elle offre la possibilité à un agent de se singulariser, de se détacher de sa classe d'appartenance au travers d'une ou plusieurs méthodes dont le code va lui être propre. Considérons par exemple un univers virtuel constitué d'une route et de voitures faisant la course sur cette route. Les voitures peuvent être des instances d'une même classe fournie par le concepteur de cet univers. Une autre personne arrivant dans cet univers peut observer l'évolution des véhicules et la juger pas assez performante. Il a alors la possibilité, sans arrêter la simulation, de choisir une voiture, de proposer pour cette voiture seulement un nouvel algorithme de conduite et d'observer comment son véhicule se comporte par rapport aux autres. Si la nouvelle approche est jugée meilleure, l'algorithme de pilotage peut alors passer du statut de méthode d'une instance au statut de méthode de classe afin que toutes les instances en profitent.

En résumé, les aspects dynamiques du langage oRis sont parmi les points importants de notre approche. Ils permettent à toute personne souhaitant interagir avec un univers virtuel, d'apporter ses propres objets, ses propres solutions, sans que cela n'ait besoin d'être prévu au préalable par le concepteur de cet univers.

4.4. *Persistance*

Comme nous l'avons vu lors des sections précédentes, ARéVi permet la création et la modification interactive d'un univers virtuel. Ces opérations conduisent à la modification (attributs + code) ou à la création de nouveaux agents. Lorsque l'utilisateur le souhaite, ou lors de l'arrêt d'une simulation, ARéVi est en mesure de générer automatiquement un fichier script en oRis contenant l'état exact du système au moment de la sauvegarde (classes, agents, sur-définition des méthodes pour certains agents etc). Le redémarrage d'une session ARéVi à partir de ce fichier permet de se retrouver dans l'état exact où l'on était au moment de l'arrêt.

4.5. *Evolution*

Alors qu'ARéVi est fondamentalement une plate-forme distribuée, le système que nous venons de décrire ne fonctionne actuellement que si la totalité des agents oRis "vivants" sont présents sur une seule CPU. Plusieurs utilisateurs peuvent néanmoins interagir et coopérer avec la simulation depuis des postes déportés grâce à des liaisons TCP véhiculant des messages oRis. La prochaine version, en cours de réalisation, sera complétement répartie, les agents pouvant alors vivre sur n'importe lequel des postes participant à la simulation.

5. Conclusion

Les premières versions d'ARéVi permettaient de réaliser des sessions 3D coopératives sur un réseau local. Les participants à ces sessions partageaient un même univers 3D, et pouvaient y coopérer en agissant sur les objets partagés de ces univers. Seules les entités de base pouvaient migrer d'une session ARéVi vers une autre (la migration d'agents n'était possible que si les agents étaient connus au niveau du noyau d'ARéVi,

c'est-à-dire qu'il fallait recompiler le noyau pour autoriser la migration de nouveaux agents).

Les problèmes qui restaient posés étaient donc essentiellement l'accès au réseau internet et la migration des agents d'une session ARéVi vers une autre.

Grâce à l'avènement du multicast et grâce à l'intégration d'un véritable langage de programmation permettant une évolution dynamique d'une session ARéVi (oRis), ARéVi devrait bientôt permettre de réaliser des sessions coopératives entre des sites distants, et des agents évolués pourront migrer d'un site vers un autre.

Références

[1] Balaguer J.F. *Virtual Studio : Un système d'animation en environnement virtuel*. PhD thesis, EPFL DI–LIG, 1993.

[2] Brutzman D.P. *A virtual world for an autonomous underwater vehicle*. PhD thesis, Naval Postgraduate School, Monterey, California, 1994.

[3] Carlsson C. and Hagsang O. Dive – a platform for multi–user virtual environnement. *Computer and Graphics*, pages 663–669, 1993.

[4] Codella C.F, Jalili R., Koved L. , and Lewis J.B. A toolkit for developing multi-user, distributed virtual environnements. In *IEEE Virtual Reality Annual International Symposium*, 1993.

[5] Gobbetti E. *Virtuality Builder II, vers une architecture pour l'interaction avec des mondes synthétiques*. PhD thesis, EPFL DI–LIG, 1993.

[6] Green M., Shaw C., and White L. Minimal reality toolkit, version 1.3. Technical report, Department of Computing Science, University of Alberta, 1993.

[7] Harrouet F., Cozien R., Reignier P., and Tisseau J. oris : un langage de simulation multi-agents. In *JFIADSMA'97*, Avril 1997.

[8] Hartman J. and Wernecke J. *The VRML 2.0 Handbook : Building Moving Worlds on the Web*. Addison-Wesley Publishing Company, 1996.

[9] Macedonia M.R. *A Network software Architecture For Large Scale Virtual Environnements*. PhD thesis, Naval Postgraduate School, Monterey, California, 1995.

[10] Roohlf J. and Helman J. Iris performer : A high performance multiprocessing toolkit for real–time 3d graphics. In *ACM SIGGRAPH*, pages 381–393, 1994.

[11] Strauss P.S. and Carey R. An object-oriented 3d graphics toolkit. In *ACM SIGGRAPH*, pages 341–347, 1992.

[12] Zeleznik R.C., Conner D.B., Wlocka M.M., Aliaga D.G., Wang N.T., Hubbard P.M., Knepp B., Kaufman H., Hugues J.F., and van Dam A. An object-oriented framework for the integration of interactive animation techniques. In *ACM SIGGRAPH*, pages 105–112, 1991.